# Intro to ML: Lecture 2

Multiple Linear Regression + Feature Transformations + Model Selection

Prof. Gustavo Sandoval

## course admin

- First lab assignment `lab_housing_partial.ipynb` due **tonight, by midnight.**
- First written assignment due **next Monday Feb 10, by midnight.**
  - $10\%$ extra credit if you use LaTeX or Markdown to typeset your assignment.

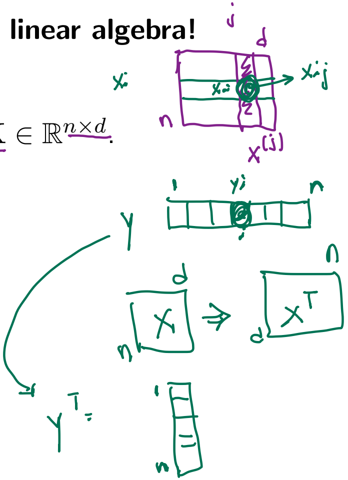
The problem set is challenging. I expect working through the problems to be one of the major ways you master material for the course. Please try to get started ASAP so that you can take advantage of office hours next week if needed.

## linear algebra review

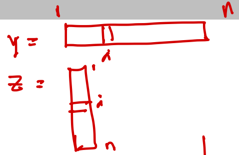
**Now it the time to review your linear algebra!**

**Notation:**

- Let $\mathbf{X}$ be an $n \times d$ matrix. Written $\mathbf{X} \in \mathbb{R}^{n \times d}$.
- $\mathbf{x}_i$ is the $i^{\text{th}}$ row of the matrix. ✓
- $\mathbf{x}^{(j)}$ is the $j^{\text{th}}$ column.
- $x_{ij}$ is the $i, j$ entry.
- For a vector $\mathbf{y}$, $y_i$ is the $i^{\text{th}}$ entry.
- $\mathbf{X}^T$ is the matrix transpose.
- $\mathbf{y}^T$ is a vector transpose.



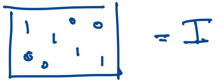
## linear algebra review



**Things to remember:**

- Matrix multiplication. If I multiply $\mathbf{X} \in \mathbb{R}^{\underline{n} \times \boxed{d}}$ by $\mathbf{Y} \in \mathbb{R}^{\boxed{d} \times \underline{k}}$ I get $\mathbf{XY} = \mathbf{Z} \in \mathbb{R}^{\underline{n \times k}}$.
- Inner product/dot product. $\langle \mathbf{y}, \mathbf{z} \rangle = \sum_{i=1}^{n} \underline{y_i z_i}$.
- $\langle \mathbf{y}, \mathbf{z} \rangle = \mathbf{y}^T \mathbf{z} = \mathbf{z}^T \mathbf{y}$.
- Euclidean norm (L2): $\|\mathbf{y}\|_2 = \sqrt{\mathbf{y}^T \mathbf{y}}$.
- $\|\mathbf{y}\|_2^2 = \langle \mathbf{y}, \mathbf{y} \rangle = \mathbf{y}^T \mathbf{y}$.
- $(\mathbf{XY})^T = \mathbf{Y}^T \mathbf{X}^T$.

## linear algebra review



**Things to remember:**

- Identity matrix is denoted as $\mathbf{I}$.
- "Most" square matrices have an inverse: i.e. if $\underline{\mathbf{Z}} \in \mathbb{R}^{n\times n}$, there is a matrix $\mathbf{Z}^{-1}$ such that $\mathbf{Z}^{-1}\mathbf{Z} = \mathbf{Z}\mathbf{Z}^{-1} = \mathbf{I}$.
- Let $\mathbf{D} = \text{diag}(\mathbf{d})$ be a diagonal matrix containing the entries in $\mathbf{d}$.
- $\mathbf{XD}$ scales the columns of $\mathbf{X}$. $\mathbf{DX}$ scales the rows.

## linear algebra review

You also need to be comfortable working with matrices in `numpy` .
Go through the `Matrices Demo` slowly.

## REMINDER: supervised regression

**Training Dataset:**

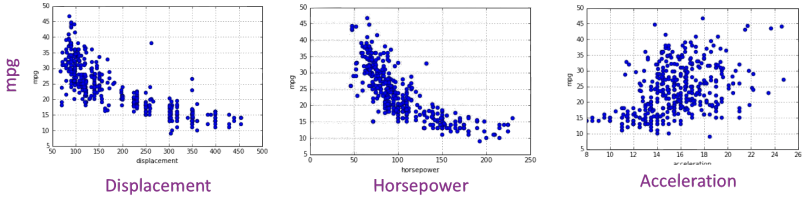
- Given input pairs $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)$.
- Each $\mathbf{x}_i$ is an input data vector (the predictor).
- Each $y_i$ is a continuous output variable (the target).

**Objective:**

- Have the computer automatically find some function $f(\mathbf{x})$ such that $f(\mathbf{x}_i)$ is close to $y_i$ for the input data.

## example from last class

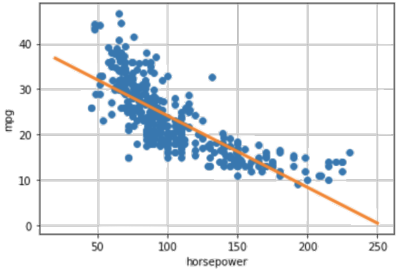
Predict miles per gallon of a vehicle given information about its engine/make/age/etc.



## example from last class

**Dataset:**

- $x_1, \ldots, x_n \in \mathbb{R}$ (horsepowers of $n$ cars – this is the predictor/independent variable)
- $y_1, \ldots, y_n \in \mathbb{R}$ (MPG – this is the response/dependent variable)



## supervised learning definitions

**What are the three components needed to setup a supervised learning problem?**

- **Model** $f_{\boldsymbol{\theta}}(x)$**:** Class of equations or programs which map input $x$ to predicted output. We want $f_{\boldsymbol{\theta}}(x_i) \approx y_i$ for training inputs.
- **Model Parameters** $\boldsymbol{\theta}$**:** Vector of numbers. These are numerical nobs which parameterize our class of models.
- **Loss Function** $L(\boldsymbol{\theta})$**:** Measure of how well a model fits our data. Typically some function of $f_{\boldsymbol{\theta}}(x_1) - y_1, \ldots, f_{\boldsymbol{\theta}}(x_n) - y_n$

**Empirical Risk Minimization:** Choose parameters $\boldsymbol{\theta}^*$ which minimize the Loss Function:

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} L(\boldsymbol{\theta})$$

simple linear regression

## Simple Linear Regression

- **Model**: $f_{\beta_0,\beta_1}(x) = \beta_0 + \beta_1 \cdot x$

$$\overset{\text{predicted}}{\hat{y}_i} = f_{\beta_0 \beta_1}(x_i)$$

- **Model Parameters**: $\beta_0, \beta_1$

- **Loss Function**: $L(\beta_0, \beta_1) = \sum_{i=1}^{n} (\overset{\text{actual}}{y_i} - f_{\beta_0,\beta_1}(x_i))^2$

$$(y_i - \hat{y}_i)^2$$

**Goal:** Choose $\beta_0, \beta_1$ to minimize
$L(\beta_0, \beta_1) = \sum_{i=1}^{n} |y_i - \beta_0 - \beta_1 x_i|^2$.

## multiple linear regression

### Multiple Linear Regression

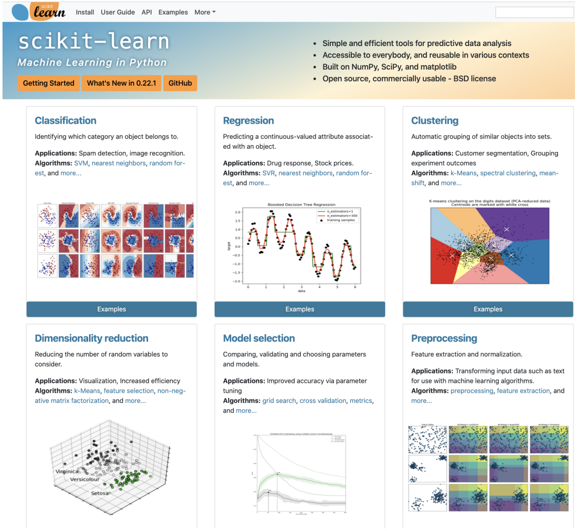
Predict target $y$ using **multiple features**, simultaneously.
**Motivating example:** Predict diabetes progression in patients after 1 year based on health metrics. (Measured via numerical score.)

**Features:** Age, sex, body mass index, average blood pressure, six blood serum measurements (e.g. cholesterol, lipid levels, iron, etc.)

Demo in `demo_diabetes.ipynb`.

## libraries for this demo

Introducing **Scikit Learn**.



Install User Guide API Examples More

scikit-learn

*Machine Learning in Python*

Getting Started What's New in 0.22.1 GitHub

- Simple and efficient tools for predictive data analysis
- Accessible to everybody, and reusable in various contexts
- Built on NumPy, SciPy, and matplotlib
- Open source, commercially usable - BSD license

**Classification**

Identifying which category an object belongs to.

**Applications:** Spam detection, image recognition.
**Algorithms:** SVM, nearest neighbors, random forest, and more...

Examples

**Regression**

Predicting a continuous-valued attribute associated with an object.

**Applications:** Drug response, Stock prices.
**Algorithms:** SVR, nearest neighbors, random forest, and more...

Examples

**Clustering**

Automatic grouping of similar objects into sets.

**Applications:** Customer segmentation, Grouping experiment outcomes
**Algorithms:** k-Means, spectral clustering, mean-shift, and more...

Examples

**Dimensionality reduction**

Reducing the number of random variables to consider.

**Applications:** Visualization, Increased efficiency
**Algorithms:** k-Means, feature selection, non-negative matrix factorization, and more...

**Model selection**

Comparing, validating and choosing parameters and models.

**Applications:** Improved accuracy via parameter tuning
**Algorithms:** grid search, cross validation, metrics, and more...

**Preprocessing**

Feature extraction and normalization.

**Applications:** Transforming input data such as text for use with machine learning algorithms.
**Algorithms:** preprocessing, feature extraction, and more...

Boosted Decision Tree Regression
n_estimators=1
n_estimators=300
training samples
K-means clustering on the digits dataset (PCA-reduced data)
Centroids are marked with white cross
Virginica
Versicolour
Setosa

## scikit learn

**Pros:**

- One of the most popular “traditional” ML libraries.
- Many built in models for regression, classification, dimensionality reduction, etc.
- Easy to use, works with ‘numpy‘, ‘scipy‘, other libraries we use.
- Great for rapid prototyping, testing models.

**Cons:**

- Everything is very “black-box”: difficult to debug, understand why models aren’t working, speed up code, etc.

## scikit learn

**Modules used:**

- `datasets` module contains a number of pre-loaded datasets. Saves time over downloading and importing with `pandas`.
- `linear_model` can be used to solve Multiple Linear Regression. A bit overkill for this simple model, but gives you an idea of `sklearn`'s general structure.

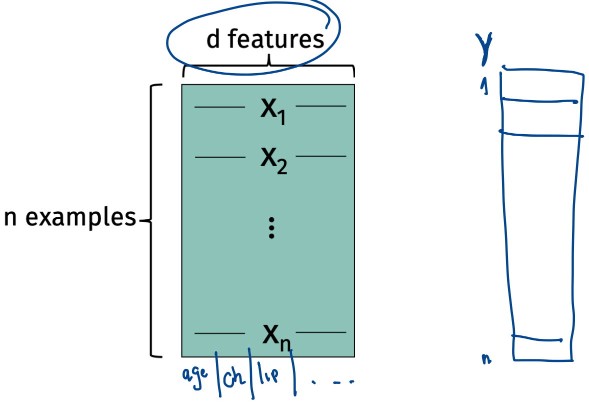
## the data matrix

**Target variable:**

- Scalars $y_1, \ldots, y_n$ for $n$ data examples (a.k.a. samples).

**Predictor variables:**

- $d$ dimensional vectors $\mathbf{x}_1, \ldots, \mathbf{x}_n$ for $n$ data examples and $d$ features



## the data matrix

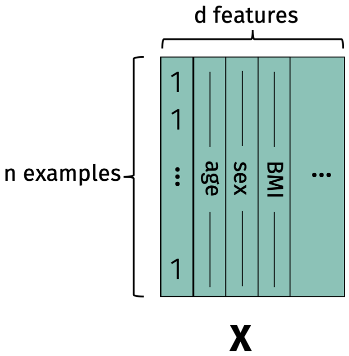
**Target variable:**

- Scalars $y_1, \ldots, y_n$ for $n$ data examples (a.k.a. samples).

**Predictor variables:**

- $d$ dimensional vectors $\mathbf{x}_1, \ldots, \mathbf{x}_n$ for $n$ data examples and $d$ features

$Y_i = \cancel{\beta_0} + \beta_1 X_i$



## multiple linear regression

**Data matrix indexing:**

$$\mathbf{X} = \begin{bmatrix} x_{11} & x_{12} & \dots & x_{1d} \\ x_{21} & x_{22} & \dots & x_{2d} \\ x_{31} & x_{32} & \dots & x_{3d} \\ \vdots & \vdots & & \vdots \\ x_{n1} & x_{n2} & \dots & x_{nd} \end{bmatrix}$$

**Multiple Linear Regression Model:**

Predict $y_i \approx \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \ldots + \beta_d x_{id}$

The rate at which diabetes progresses depends on many factors, with each factor having a different magnitude effect.

## matrix form of linear regression

- Predicted value for the $i$th example is given by:
  $\hat{y}_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \ldots + \beta_d x_{id}$
- Define feature matrix and regression vector:

$$\underset{n \times d}{\mathbf{X}} = \begin{bmatrix} x_{11} & x_{12} & \ldots & x_{1d} \\ x_{21} & x_{22} & \ldots & x_{2d} \\ x_{31} & x_{32} & \ldots & x_{3d} \\ \vdots & \vdots & & \vdots \\ x_{n1} & x_{n2} & \ldots & x_{nd} \end{bmatrix} \quad \underset{d+1 \times 1}{\beta} = \begin{bmatrix} \underline{\beta_0} \\ \beta_1 \\ \beta_2 \\ \vdots \\ \beta_d \end{bmatrix}$$

- Predicted values are given by: $\boxed{\hat{\mathbf{y}}} = \underline{\mathbf{X}\beta}$
- Given a new example $\mathbf{x}$, we can predict the target value $\hat{y}(\mathbf{x}) = [\underline{1}, \mathbf{x}^{\mathbf{T}}]\beta$

## multiple linear regression

**Assume first columns contains all $1$'s.** If it doesn't append on a column of all $1$'s.

$$\mathbf{X} = \begin{bmatrix} x_{11} & x_{12} & \dots & x_{1d} \\ x_{21} & x_{22} & \dots & x_{2d} \\ x_{31} & x_{32} & \dots & x_{3d} \\ \vdots & \vdots & & \vdots \\ x_{n1} & x_{n2} & \dots & x_{nd} \end{bmatrix} = \begin{bmatrix} 1 & x_{12} & \dots & x_{1d} \\ 1 & x_{22} & \dots & x_{2d} \\ 1 & x_{32} & \dots & x_{3d} \\ \vdots & \vdots & & \vdots \\ 1 & x_{n2} & \dots & x_{nd} \end{bmatrix}$$

**Multiple Linear Regression Model:**

Predict $y_i \approx \beta_1 x_{i1} + \beta_2 x_{i2} + \ldots + \beta_d x_{id}$

## multiple linear regression

**Use as much linear algebra notation as possible!**

- Model: $f(x) = \hat{y}_i = \beta_1 x_{i1} + \beta_2 x_{i2} \ldots \beta_d x_{id}$
  $\sum_{i=1}^{d} \beta_i x_i = \langle \beta, x \rangle$
- Model Parameters: $[\beta_1, \beta_2 \ldots \beta_d] = \vec{\beta}$
- Loss Function: $L_\beta = |y_i - \hat{y}_i|^2$
  $= |y_i - (\beta_1 x_{i1} \ldots \beta_d x_{id})|^2$
  $\|y - \langle \beta, x \rangle\|_2$

## multiple linear regression

**Linear Least-Squares Regression.**

- Model:

$$f_{\boldsymbol{\beta}}(\mathbf{x}) = \langle \mathbf{x}, \boldsymbol{\beta} \rangle$$

- Model Parameters:

$$\boldsymbol{\beta} = [\beta_1, \beta_2, \ldots, \beta_d]$$

- Loss Function:

$$\begin{aligned} L(\boldsymbol{\beta}) &= \sum_{i=1}^{n} |y_i - \langle \mathbf{x}_i, \boldsymbol{\beta} \rangle|^2 \\ &= \|\mathbf{y} - \mathbf{X}\beta\|_2^2 \end{aligned}$$

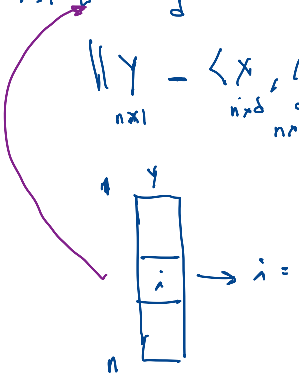
## linear algebraic form of loss function

$$\sum_{i=1}^{n} \left[ y_i - \beta_1 x_{i1} + \beta_2 x_{i2} \ldots \beta_d x_{id} \right]$$

$d$

$X = n \times d$

$$\Big\| \underset{n\times 1}{Y} - \underset{n\times 1}{\langle \underset{n\times d}{X}, \underset{d\times 1}{\beta} \rangle} \Big\|_2^2$$



## loss minimization

**Machine learning goal:** minimize the loss function
$L(\boldsymbol{\beta}) : \mathbb{R}^d \to \mathbb{R} = \|\mathbf{y} - \mathbf{X}\beta\|_2^2$.

Find optimum by determining for which $\boldsymbol{\beta} = [\beta_1, \ldots, \beta_d]$ all partial derivatives are $0$. I.e. when do we have:

$$\begin{bmatrix} \frac{\partial L}{\partial \beta_1} \\ \frac{\partial L}{\partial \beta_2} \\ \vdots \\ \frac{\partial L}{\partial \beta_d} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

## the all important gradient

For any function $L(\boldsymbol{\beta}) : \mathbb{R}^d \to \mathbb{R}$, the gradient $\nabla L(\beta)$ is a function from $\mathbb{R}^d \to \mathbb{R}^d$ defined:

$$\nabla L(\beta) = \begin{bmatrix} \frac{\partial L}{\partial \beta_1} \\ \frac{\partial L}{\partial \beta_2} \\ \vdots \\ \frac{\partial L}{\partial \beta_d} \end{bmatrix}$$

The gradient of the loss function is a central tool in machine learning. We will use it again and again.

## gradient

**Loss function:**

$$L(\boldsymbol{\beta}) = \|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|_2^2$$

**Gradient:**

$$-2 \cdot \mathbf{X}^T(\mathbf{y} - \mathbf{X}\boldsymbol{\beta})$$

Find optimum by determining for which $\boldsymbol{\beta} = [\beta_1, \ldots, \beta_d]$ the gradient is $0$. i.e. when do we have:

$$\nabla L(\beta) = \begin{bmatrix} \frac{\partial L}{\partial \beta_1} \\ \frac{\partial L}{\partial \beta_2} \\ \vdots \\ \frac{\partial L}{\partial \beta_d} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

## loss minimization

**Goal:** minimize the loss function $L(\boldsymbol{\beta}) = \|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|_2^2$.

$$\begin{aligned}\nabla L(\boldsymbol{\beta}) &= -2 \cdot \mathbf{X}^T(\mathbf{y} - \mathbf{X}\boldsymbol{\beta}) \\ &= 2\mathbf{X}^T\mathbf{X}\boldsymbol{\beta} - 2\mathbf{X}^T\mathbf{y} = \mathbf{0}\end{aligned}$$

**Solve for optimal $\beta^*$:**

$$\begin{aligned}\mathbf{X}^T\mathbf{X}\boldsymbol{\beta}^* &= \mathbf{X}^T\mathbf{y} \\ \boldsymbol{\beta}^* &= \left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{y}\end{aligned}$$

## multiple linear regression solution

Need to compute $\boldsymbol{\beta}^* = \arg\min_{\boldsymbol{\beta}} \|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|_2^2 = \left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{y}$.

- Main cost is computing $(\mathbf{X}^T\mathbf{X})^{-1}$ which takes $O(nd^2)$ time.
- Can solve slightly faster using the method `numpy.linalg.lstsq`, which is running an algorithm based on QR decomposition.
- For larger problems, can solve much faster using an *iterative methods* like `scipy.sparse.linalg.lsqr`.

Will learn more about iterative methods when we study Gradient Descent.

## gradient warmup

**Function:**

$$g(\mathbf{z}) = \mathbf{a}^T\mathbf{z} \text{ for some fixed vector } \mathbf{a} \in \mathbb{R}^d$$

**Gradient:**

$\Delta g(z) = \begin{bmatrix} \partial g/\partial z_1 \\ \partial g/\partial z_2 \\ \vdots \\ \partial g/\partial z_d \end{bmatrix}$ recall that $\partial g/\partial z_1 = a_1 + \cancel{a_2 z_2} \ldots \cancel{a_d z_d}$

$\Rightarrow \Delta g(z) \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_d \end{bmatrix} = a$

**Function:**

$$f(\mathbf{z}) = \|\mathbf{z}\|_2^2$$

**Gradient:**

here the same $\nabla f(z) = 2z$

because $\frac{\partial f}{\partial z_i} \|z\|_2^2 = \partial/\partial z_i \sum_{i=1}^{d} z_i^2 = 2z_i$

## test your intuition

$$y = \beta_0 + \beta_1 \underset{\uparrow}{X}$$

**Example from book:** What is the sign of $\beta_1$ when we run a simple linear regression using the following predictors for number of sales in a particular market as a function of:

- Amount of TV advertising in that market: $+$
- Amount of print advertising in that market: $+$

## interacting variables

What is the sign of the corresponding $\beta$'s when we run a multiple linear regression using the following predictors together:

- Amount of TV advertising in that market: Positive
- Amount of print advertising in that market: Negative, close to zero

**Can you explain this? Try to think of your own example of a regression problem where this phenomenon might show up.**

## dealing with categorical variables

The sex variable in the diabetes problem was binary. We encoded it as 2 numbers – e.g. (0,1), (-1,1), (1,2).
Suppose we go back to the MPG prediction problem. What if we had a categorical predictor variable for car make with more than 2 options: e.g. Ford, BMW, Honda. **How would you encode as a numerical column?**

$$\begin{bmatrix} \texttt{ford} \\ \texttt{ford} \\ \texttt{honda} \\ \texttt{bmw} \\ \texttt{honda} \\ \texttt{ford} \end{bmatrix} \rightarrow \begin{bmatrix} 1 \\ 1 \\ 2 \\ 3 \\ 2 \\ 1 \end{bmatrix}$$

## one hot encoding

**Better approach:** One Hot Encoding.

$$\begin{bmatrix} \texttt{ford} \\ \texttt{ford} \\ \texttt{honda} \\ \texttt{bmw} \\ \texttt{honda} \\ \texttt{ford} \end{bmatrix} \rightarrow \begin{bmatrix} \bigcirc\!\!\!\!1 & 0 & 0 \\ \bigcirc\!\!\!\!1 & 0 & 0 \\ 0 & \bigcirc\!\!\!\!1 & 0 \\ 0 & 0 & \bigcirc\!\!\!\!1 \\ 0 & \bigcirc\!\!\!\!1 & 0 \\ \bigcirc\!\!\!\!1 & 0 & 0 \end{bmatrix}$$

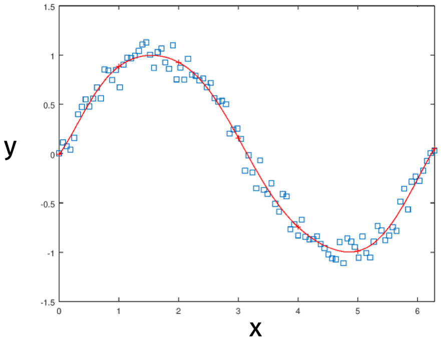
- Create a separate feature for every category, which is 1 when the variable is in that category, zero otherwise.
- Not too hard to do by hand, but you can also use library functions like `sklearn.preprocessing.OneHotEncoder`.

**Avoids adding inadvertent linear relationships.**

## transformed linear models

Suppose we have singular variate data examples $(x, y)$. How could we fit the non-linear model:

$$y \approx \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3.$$



## transformed linear models

Transform into a multiple linear regression problem:

$$\mathbf{X} = \begin{bmatrix} 1 & x_1 & x_1^2 & x_1^3 \\ 1 & x_2 & x_2^2 & x_2^3 \\ 1 & x_3 & x_3^2 & x_3^3 \\ \vdots & \vdots & & \vdots \\ 1 & x_n & x_n^2 & x_n^3 \end{bmatrix}$$

Each column $j$ is generated by a different basis function $\phi_j(x)$. Could have:

- $\phi_j(x) = x^q$
- $\phi_j(x) = sin(x)$
- $\phi_j(x) = cos(10x)$
- $\phi_j(x) = 1/x$

## transformed linear models

**Transformations can also be for multivariate data.**

**Example:** Multinomial model.

- Given a dataset with target $y$ and predictors $x, z$.
- For inputs $(x_1, z_1), \ldots, (x_n, z_n)$ construct the data matrix:

$$\begin{bmatrix} 1 & x_1 & x_1^2 & z_1 & z_1^2 & x_1 z_1 \\ 1 & x_2 & x_2^2 & z_2 & z_2^2 & x_2 z_2 \\ \vdots & \vdots & & \vdots & & \\ 1 & x_n & x_n^2 & z_n & z_n^2 & x_n z_n \end{bmatrix}$$

- Captures non-linear interaction between $x$ and $y$.

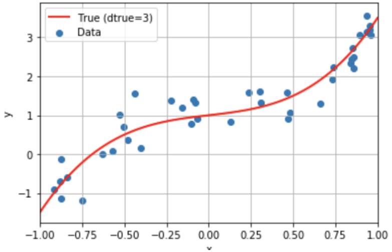
## model selection

**Remainder of lecture:** Learn about model selection, test/train paradigm, and cross-validation through a simple example.

## fitting a polynomial

**Simple experiment:**

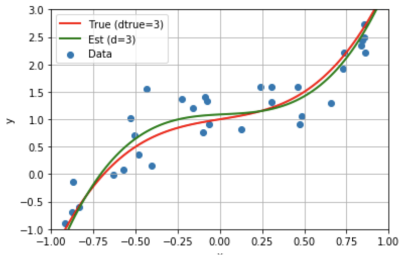
- Randomly select data points $x_1, \ldots, x_n \in [-1, 1]$.
- Choose a degree 3 polynomial $p(x)$.
- Create some fake data: $y_i = p(x_i) + \eta$ where $\eta$ is a random number (e.g random Gaussian).



## fitting a polynomial

**Simple experiment:**

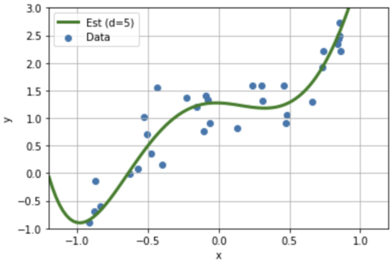
- Use multiple linear regression to fit a degree $3$ polynomial.

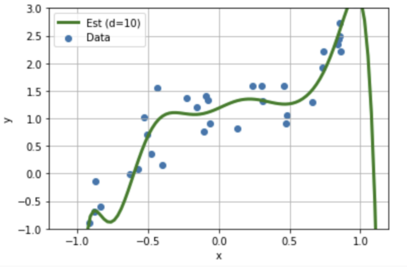


## fitting a polynomial

**What if we fit a higher degree polynomial?**

- Fit degree $5$ polynomial under squared loss.
- Fit degree $10$ polynomial under squared loss.

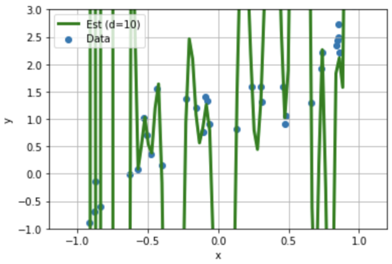




## fitting a polynomial

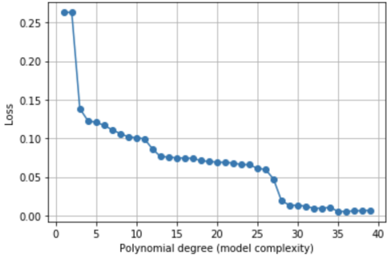
**Even higher?**

- Fit degree $40$ polynomial under squared loss.



## model selection

The more **complex** our model class (i.e. the higher degree we allow) the better our loss:

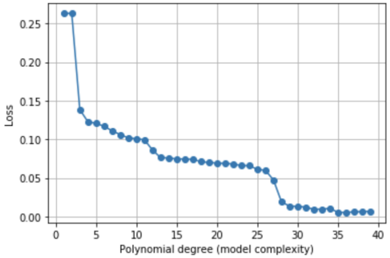


**Is our model getting better and better?**

**Given the raw data, how do we know which model to choose? Degree 3? Degree 5? Degree 40?**

## model selection

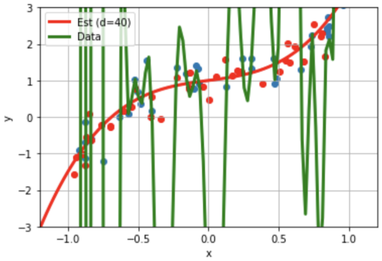
The more **complex** our model class the better our loss:



**So training loss alone is not usually a good metric for model selection. Small loss does not imply generalization.**

## model selection

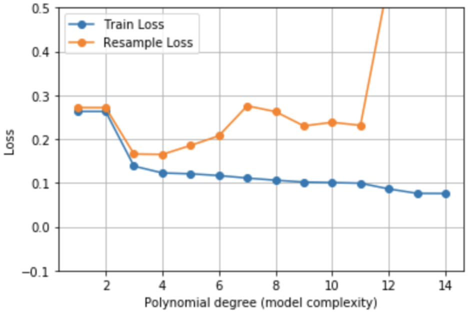
**Problem:** Loss alone is not informative for choosing model.
For more complex models, we get smaller loss on the training data, but don't expect to perform well on "new" data:



In other words, the model does not **generalize.**

## model selection

**Solution:** Directly test model on “new data”.



- Loss continues to decrease as model complexity grows.
- Performance on new data “turns around” once our model gets too complex. Minimized around degree $4$.

## train-test paradigm

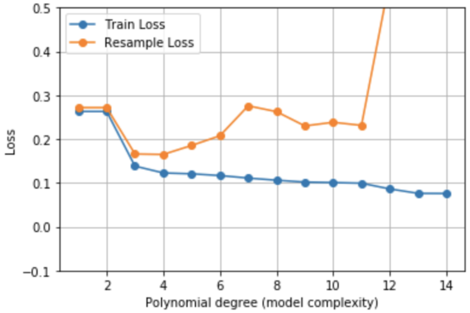
**Better approach:** Evaluate model on fresh test data which was not used during training.

**Test/train split:**

- Given data set $(\mathbf{X}, \mathbf{y})$, split into two sets $(\mathbf{X}_{\text{train}}, \mathbf{y}_{\text{train}})$ and $(\mathbf{X}_{\text{test}}, \mathbf{y}_{\text{test}})$.
- Train $q$ models $f^{(1)}, \ldots, f^{(q)}$ by finding parameters which minimize the loss on $(\mathbf{X}_{\text{train}}, \mathbf{y}_{\text{train}})$.
- Evaluate loss of each trained model on $(\mathbf{X}_{\text{test}}, \mathbf{y}_{\text{test}})$.

Sometimes you will see the term **validation set** instead of test set. Sometimes there will be both: use validation set for choosing the model, and test set for getting a final performance measure.

## train-test paradigm



- **Train loss** continues to decrease as model complexity grows.
- **Test loss** “turns around” once our model gets too complex. Minimized around degree $3 - 4$.

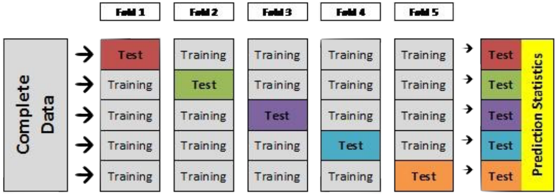
## generalization error

If the test loss remains low, we say that the model **generalizes**.
Test lost is often called **generalization error.**

## train-test paradigm

**Typical train-test split:** 70-90% / 10-30%. Trade-off between between optimization of model parameters and better estimate of model performance.

## k-fold cross validation



- Randomly divide data in $K$ parts.
  - Typical choice: $K = 5$ or $K = 10$.
- Use $K - 1$ parts for training, $1$ for test.
- For each model, compute test loss $L_{ts}$ for each "fold".
- Choose model with best average loss.
- Retrain best model on entire dataset.

## k-fold cross validation

**Leave-one-out cross validation**: take $K = n$, where $n$ is our total number of samples.

**Is there any disadvantage t o choosing $K$ larger?**